INF-253 Lenguajes de Programación

Tarea 2: C

21 de agosto de 2023

1. TreasureFinder

Mares de Muc Haagua, año 30XX. Nos encontramos en guerra con un centenar de barcos piratas, todos buscan lo mismo, los tesoros dentro del bosque de Muc Haagua. Es muy peligroso mostrarse directamente al oponente, así que lo mejor que podemos hacer es seguir el ejemplo del dispositivo milenario: Battleship™. Según nuestro conocimiento, las sociedades de épocas pasadas lo usaban como una especie de simulación extremadamente fiel a la realidad de las guerras marítimas. Nuestro sabelotodo Kno Witall, cree que ™ era una abreviación de Tactician's Machine y dicen que aquel que logre masterizar el arte de "achuntarle" tendrá poder sobre todas las riquezas del mundo.

~ Capitán Har Tabarba

Bosque de Muc Haagua, ehhh… ¿qué día es hoy?, no importa, pocas noches después de la bitácora anterior. Ganamos la batalla "achuntándole" a las naves enemigas y entramos al territorio. Estamos en busca del tesoro, pero parece ser que todo se encuentra bajo tierra y no existe forma de saber dónde podría haber riquezas. Tal parece que volveremos a depender de Battleship: Tactitian's Machine, tenemos suficientes bombas para actuar, pero el tiempo es corto, en cualquier momento llegarán otros piratas.

~ Capitán Har Tabarba

FIN

Parece que Har Tabarba necesita ayuda encontrando los tesoros, por lo que pide a sus ingenieros crear un mecanismo para colocar bombas en zonas selectas del bosque. El programa que le solicitan, *TreasureFinder*, parece un juego, y podría serlo, pero es vital que aplique sus conocimientos de punteros en *C* para ayudar al pirata a llevarse todo el tesoro que pueda (Y que le entreguen a usted su parte de la ganancia).

2. El juego

TreasureFinder consiste en una variación del juego de mesa *Battleship*, donde uno dispara misiles en un tablero y espera derribar las naves del enemigo. En esta versión, el jugador deja distintos tipos de bomba en el tablero, para que cada vez que exploten se excaven ciertas celdas en búsqueda de los tesoros.

El tablero del juego se selecciona al inicio del programa, donde se puede elegir entre los tamaños 7x7, 10x10 y 12x12.

2.1 Tierra

Dentro de cada celda del tablero se encontrará un bloque de tierra de la siguiente forma:

```
Typedef struct Tierra{
  int vida;
  int es_tesoro;
}
```

Donde *vida* es un entero que va desde 0 a 3, simbolizando cuántas explosiones puede recibir el bloque de tierra hasta que revele si esconde un tesoro o no y *es_tesoro* es un entero que acepta valores 0 y 1, simbolizando si el bloque de tierra esconde o no un tesoro.

2.2 Bombas

Las bombas se representarán como estructuras que guardarán los siguientes datos:

```
Typedef struct Bomba{
  int contador_turnos;
  void (*explotar)(int fila, int columna);
  Tierra* tierra_debajo;
}
```

Donde *contador_turnos* guarda la cuenta de cuántos turnos quedan para que la bomba explote, *explotar* es un puntero a una función que se asigna al crearse la Bomba y *tierra_debajo* es un puntero a un *struct Tierra*, que será ocupado cuando se coloque una *Bomba* en una celda donde ya había *Tierra*.

En la sección **Funciones a Implementar** se verá en profundidad cómo deberían explotar las bombas, además de cómo cada tipo afecta a *contador_turnos*.

3. Flujo del Juego

El programa inicia preguntándole al usuario qué tamaño de tablero desea jugar. Luego de escoger, el juego generará el tablero, empezando con poner los tesoros en posiciones al azar y rellenando todo lo demás con tierra. Con esto listo, parte el primer turno del juego.

Durante el juego, cada turno sigue el siguiente flujo:

1. Cada bomba en el tablero reduce su contador en 1. Si el contador de una bomba llega a 0, explota.
2. El juego muestra el estado del tablero al jugador, indicando las celdas con tierra, bombas y tesoros descubiertos. (Tierra: {3, 2, 1, 0}, Bomba: o, Tesoro: *)
3. En pantalla se muestran las opciones de juego, de las cuales el jugador debe elegir 1, ejecutándola.
4. Termina el turno cuando el jugador coloca una bomba.

Si luego de una explosión se revelan todos los tesoros del tablero, el juego termina, notificando al jugador de su victoria y cerrando el programa luego de cualquier input.

A continuación, se muestra un ejemplar bien detallado de cómo se vería el flujo del juego en el terminal.

```
¡Bienvenido a TreasureFinder!
Indique el tamaño del tablero a jugar:
1.7x7  2.10x10  3.12x12
Su input: 1
Empezando juego... ¡listo!
Tablero (Turno 1)
2 | 2 | 1 | 3 | 1 | 2 | 3
3 | 2 | 1 | 1 | 2 | 1 | 3
3 | 1 | 3 | 2 | 2 | 1 | 1
2 | 2 | 1 | 2 | 3 | 2 | 3
1 | 2 | 2 | 1 | 3 | 2 | 2
2 | 3 | 2 | 2 | 1 | 3 | 1
2 | 1 | 2 | 1 | 3 | 2 | 1
Seleccione una accion:
1.Colocar Bomba  2.Mostrar Bombas  3.Mostrar Tesoros
Escoja una opcion: 1
Indique coordenadas de la bomba
Fila: 4
Columna: 6
Indique forma en que explota la bomba
1.Punto  2.X
Su input: 2

Tablero (Turno 2)
2 | 2 | 1 | 3 | 1 | 2 | 3
3 | 2 | 1 | 1 | 2 | 1 | 3
3 | 1 | 3 | 2 | 2 | 1 | 1
2 | 2 | 1 | 2 | 3 | o | 3
1 | 2 | 2 | 1 | 3 | 2 | 2
2 | 3 | 2 | 2 | 1 | 3 | 1
2 | 1 | 2 | 1 | 3 | 2 | 1
```

... (Prosigue el juego con el siguiente turno)

Se recomienda usar este mismo formato o uno similar para su tarea, en caso de usar uno distinto, procure que el jugador pueda ver la información relevante del juego para poder tomar sus decisiones, como el estado del tablero, cuándo debe colocar un input y qué opciones puede colocar, etc.

4. Funciones a Implementar

A continuación, se presentan todas las funciones que deben implementarse para la tarea:

- IniciarTablero: Recibe el tamaño del tablero *n*, generando una matriz *void n x n* inicializado con *Tierra* en cada una de sus celdas. Cada *Tierra* se le asigna al azar *vida* entre los valores {1, 2, 3} (equiprobable) además del parámetro *es_tesoro* (5% de probabilidad de ser un tesoro).
- PasarTurno: Recibe el tablero del juego. Recorre la matriz de izquierda a derecha y de arriba abajo intentando explotar cada una de las bombas en el tablero con *TryExplotar*.
- ColocarBomba: Recibe una *Bomba* y coordenadas. Coloca una *Bomba* en el tablero, en la celda especificada por el parámetro coordenadas.
- TryExplotar: Recibe coordenadas. Reduce *contador_turnos* de la *Bomba* en la coordenada en 1, si termina en 0 el contador, debe llamar la función *explotar* de la *Bomba.*
- MostrarTablero: Muestra el tablero representando la *Tierra* con su *vida*, *Bomba* con *o (*letra "o" minúscula) y un tesoro descubierto con *. (Tierra = {"3", "2", "1", "0"}, Bomba = "o", Mostrar tesoro de *Tierra* = "*". Notar que si una *Tierra* llega a *vida = 0* y no es un tesoro, simplemente se enseñará como "0")
- MostrarBombas: Muestra por pantalla cada una de las bombas en el tablero de la siguiente forma:

```
Turnos para explotar: el valor

Coordenada: x y

Forma Explosión: nombre de la función explotar (ExplosionPunto o
ExplosionX)

Vida de Tierra Debajo: el valor.
```

- Es importante notar que las coordenadas *x* e *y* van desde 1 hasta el tamaño del tablero.

- BorrarBomba: Recibe coordenadas. Borra la *Bomba* en la coordenada de la memoria *Heap*, pero antes de eso devuelve la *Tierra* al tablero.
- BorrarTablero: Borra el tablero de la memoria *Heap*.

- VerTesoros: Muestra todos los tesoros en el tablero (ocultos y revelados). Esta función está pensada para facilitar el testeo y se debe ver de la siguiente forma:

  ```
  Tesoros
  2 | * | *
  1 | 3 | 3
  2 | * | 1
  ```

Para el caso del *struct Bomba*, se deben implementar las siguientes funciones de explosión, donde cada una representa una forma de explosión y un daño a la *vida* de la *Tierra*. Para cada tipo de explosión se adjunta un ejemplo formateado de cómo se vería el tablero antes de colocar la bomba, al colocarla y después de haber explotado, en ese orden.

- ExplosionPunto: Explota la bomba como un punto, es decir, solo afecta a la tierra debajo de la Bomba. Disminuye la *vida* de esa *Tierra* en 3. Al momento de crear una bomba con este tipo de explosión, su variable *contador_turnos* debe asignarse a 1.

  Ejemplo de la explosión (Los colores son solo para notar dónde está la bomba y dónde afectó su explosión, no se considera así para el programa):

Tablero		Colocar Bomba		Explotó
3 | 3 | 3		3 | 3 | 3		3 | 3 | 3
2 | 3 | 3	→	2 | **o** | 3	→	2 | **0** | 3
1 | 3 | 2		1 | 3 | 2		1 | 3 | 2

- ExplosionX: Explota la bomba como una equis, afecta 5 celdas solamente (no se extiende por el tablero). Disminuye la vida de las celdas de *Tierra* afectadas en 1. Al momento de crear una bomba con este tipo de explosión, su variable *contador_turnos* debe asignarse a 3.

  Ejemplo:

3 | 3 | 3		3 | 3 | 3		**2** | 3 | **2**
2 | 3 | 3	→	2 | **o** | 3	→	2 | **2** | 3
1 | 3 | 2		1 | 3 | 2		**0** | 3 | **1**

Luego de explotar la *Bomba* debe ser borrada del tablero con *BorrarBomba*.

5. Sobre el Tablero

Note cómo el tablero que se dejó en el archivo *Tablero.h*, que es el que **debe** usar, es del tipo *void****. Esto quiere decir que cada celda del tablero es un puntero *void*, así, dentro del tablero puede haber tanto *Tierra* como *Bomba*. Como se ha mencionado antes, al colocar una *Bomba* en el tablero, la *Bomba* debe "almacenar" la **dirección a la *Tierra*** que había originalmente en su posición del tablero, dentro en su variable *tierra_debajo*.

Como ejemplo, si se coloca una *Bomba* en *(1, 3)*, entonces la *Tierra* que había en *(1, 3)* debe mantenerse intacta en *tierra_debajo* de *Bomba*. Ocurre lo opuesto cuando *Bomba* explota, previo a su liberación de memoria se debe devolver la *Tierra* a su lugar en el tablero.

*Hint: Como puede derivar de sus clases de punteros, el tablero void*** por su cuenta no es suficiente para acceder a los valores dentro de ella, sin embargo, otra matriz podría contener la información de qué tipos están siendo almacenados en cada celda.*

6. Detalles relevantes

- Un tesoro se descubre cuando su celda de *Tierra* tiene *vida* igual a 0.
- Al crear una *Bomba* con cierto tipo de explosión, se le debe asignar su variable c*ontador_turnos*. Para *ExplosionPunto* el valor es 1 y *X* es 3.
- Si una explosión sobrepasa el tamaño del tablero, debe dar la vuelta. Es decir, si el área de una explosión llega a la celda (**8**, 1) de un tablero 7 x 7, entonces debe explotar la celda (**1**, 1). Ocurre lo mismo con intentar explotar (**0**, 1), se pasa la explosión a (**7**, 1).
- Puede asumir que el input entregado por los usuarios siempre será correcto.
- **Las consultas se hacen en el foro disponible en la sección Tareas de aula.**
- En los archivos que se subieron junto a este Enunciado, se dejaron las definiciones de las funciones antes descritas. Se pueden agregar más funciones si se requiere, pero deben avisar sobre ellas en el README de su entrega, no informar sobre ellas implica un descuento.
- Si una explosión afecta a una celda donde se encuentra una *Bomba*, el daño se debe redirigir a la *Tierra* debajo de la *Bomba.* Pero no ocurre nada con la *Bomba* en sí.

7. Sobre la Entrega

- El código debe venir indentado y ordenado.
- Se deberá entregar un .tar.gz con los siguientes archivos:
    - Bomba.c, Bomba.h, Tablero.c, Tablero.h, Tierra.h, TreasureFinder.c
    - makefile
    - README.txt

- Las funciones deberán ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo). Se deja libertad al formato del comentario.
- Debe estar presente el archivo MAKEFILE para que se efectúe la revisión, este debe compilar TODOS los archivos.
- Se debe trabajar de forma individual.
- La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea2LP_RolAlumno.tar.gz. Ejemplo: *Tarea2LP_202200000-0.tar.gz*
- El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la compilación y utilización de su programa.
- La entrega será vía aula y el plazo máximo de entrega es hasta el 08 de septiembre de 2023 a las 23:55 hora aula.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Si el makefile no está bien realizado, la tarea no se revisará.
- Se utilizará Valgrind para detectar los leak de memoria.
- Solo se pueden hacer consultas sobre la Tarea hasta 1 día antes de la fecha de entrega.

8. Calificación

9.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

9.1.1. Entrega Mínima

Para obtener el puntaje mínimo, el alumno debe cumplir los siguientes requisitos (30 pts. Total):

- Iniciar un tablero 7x7, asignando espacio mediante memoria *Heap*. Cada celda debe apuntar a una *Tierra* con *vida* distinta de 0. No es necesario que haya tesoros.
    - El tablero debe inicializarse y borrarse a través de *malloc* y *free*, no pueden haber *leaks* de memoria.
    - Requiere *MostrarTablero, IniciarTablero* y *BorrarTablero*.
- Poder colocar una *Bomba* en una celda del tablero.
    - Para la Entrega Mínima, no es necesario usar *tierra_debajo*, debe simplemente **eliminar** la *Tierra* que estaba en la celda correspondiente.
    - En el tablero debe usarse *malloc* para colocar la *Bomba*, no pueden haber *leaks* de memoria.
    - Requiere *ColocarBomba* y *BorrarBomba.*

- Tener una Interfaz de Usuario mínima para que el usuario pueda testear el juego:
    - Implica mostrar el tablero, permitir colocar una bomba y ver el tablero luego de colocarla.

9.1.2. Entrega

Luego de cumplir con la Entrega Mínima, puede obtener más puntaje cumpliendo con los siguientes puntos (puede haber puntaje parcial por cada punto):

- Uso de *tierra_debajo* para no perder la *Tierra* que se encuentra debajo de una *Bomba*. (25 pts.)
- Las explosiones dan la vuelta al tablero en caso de escaparse de sus dimensiones. (7 pts.)
- *Tierra* puede tener tesoros en el juego. (5 pts.)
- Implementación correcta de las funciones *Explosion*, no considera que el tablero dé la vuelta al pasar de largo (13 pts. Total):
    - *TryExplotar.* (1 pt.)
    - *ExplosionPunto.* (5 pts.)
    - *ExplosionX.* (7 pts)
- Implementación correcta de las siguientes funciones (20 pts. Total):
    - *IniciarTablero* para más tamaños. (3 pts.)
    - *PasarTurno*. (3 pts.)
    - *MostrarTablero* actualizando *vida* de *Tierra* y mostrando tesoros. (3 pts.)
    - *MostrarBombas.* (5 pts.)
    - *VerTesoros.* (3 pts)
    - *BorrarTablero* para más tamaños. (3 pts.)

9.2 Descuentos

Luego de cumplir con la Entrega Mínima, puede sufrir descuentos por los siguientes motivos:

- Código no ordenado (-10 puntos)
- Código no compila (-100 puntos)
- Warning (c/u -5 puntos)
- Falta de comentarios (-10pts c/u. -30 puntos MAX)
- Por cada día o fracción de día de atraso se descontarán 20 puntos (La primera hora de atraso serán solo 10 puntos).
- Falta de README (-20 pts.)
- Falta de alguna información obligatoria en el README (-5 pts. c/u)
- Porcentaje de leak de memoria ((1 - 5)% -3 puntos (6 - 50%) -15 puntos (51 - 100 %) -30 puntos)
- Mal nombre en algún archivo entregado (-20 pts si es el .tar.gz, -5 pts c/u los demás)

En caso de existir nota negativa esta será reemplazada por un 0.